

Artificial super-Bee enhanced Colony (AsBeC) algorithm for numerical optimization with limited function evaluations

Part 1: technologies and benchmark validation

Enrico Ampellio · Luca Vassio

the date of receipt and acceptance should be inserted later

Abstract The *Artificial Bee Colony* (ABC) algorithm is a very effective, simple and robust nature-based meta-heuristic optimization procedure. The standard ABC has been recently developed and it currently raises a lot of interest and upgrading efforts, since it finds attractive applications in many fields of the scientific research. One of these application is represented by optimizations that employ very expensive numerical simulations in terms of resources. In this framework there is a strong need for methods that assure the best improvement with the shortest analyses time.

In this paper, the authors propose and recall several modifications to the standard ABC in order to improve its speed and solution accuracy for problems where function evaluations have to be limited around 10^3 . These technologies consist in enhancements of the basic structure and hybridizations with other optimization strategies. Moreover three different kinds of parallelization in the code are analysed. Each modification as well as their combinations are studied and explained; the performance of modified ABC is evaluated through extensive simulations over different analytical test functions. Moreover, standard settings for this new algorithm, called *Artificial super-Bee enhanced Colony* (AsBeC), are given in order to maintain the simplicity of ABC.

The present article explores all the technical issues involved about AsBeC, while the second part will consider a real-like application to the engineering optimal design of turbomachinery through *Computational Fluid Dynamics*, environment in which the improved algorithm was originally conceived by the authors.

Keywords Artificial Bee Colony · engineering optimization · parallelization strategy · hybridization technique

1 Introduction

The applicative field of numerical optimization is normally characterized by simulation based problems, in which computational analyses are heavily time and resource consuming. There is a strong need in the scientific community for fast techniques allowing to optimize many parameters under very few function evaluations. One reference example is represented by *Computational Fluid Dynamics* (CFD), a fundamental tool in many engineering design problems. It usually implies long simulation times and optimizations based on it often involves high dimensionality in domain space.

The most widespread techniques proposed in this context lay in the class of meta-heuristic methods. Among them, one the newest and most promising is the nature-inspired *Artificial Bee Colony* algorithm (ABC)

E. Ampellio (✉) · L. Vassio
Dipartimento di Ingegneria Meccanica e Aerospaziale, Politecnico di Torino, 10129 Torino (TO), Italy
E-mail: enrico.ampellio@polito.it

L. Vassio
E-mail: luca.vassio@polito.it

(Karaboga 2007a), which combines *Particle Swarm Optimization* theory (PSO) (Kennedy and Eberhart 1995) and *Differential Evolution* principles (DE) (Price et al. 2005). At first, swarm methods could not appear promising since a colony needs multiple function evaluation at every optimization step, without any guaranteed improvement of the solution. Although this contraindication, some researchers have proven brilliant performance for ABC (Rao et al. 2010; Bertini et al. 2013). Besides, a lot of technical dissertations, test case applications and improvement works have been done in the recent years (Bolaji et al. 2013).

Starting from ABC algorithm and limiting the total number of function evaluations, the present paper is focused on modifying the original algorithm in order to:

- Improve solution accuracy, hence obtaining better optimized configuration with the same amount of function evaluations;
- Increase speed, reaching good configurations in less time or performing at the same time more function evaluations.

To accomplish these tasks, enhancements of the basic structure and hybridizations with other optimization strategies are presented. Furthermore, three possible parallelization techniques of the new algorithm are taken into account.

All the improvements are introduced singularly as technologies applied to the standard ABC. They are especially addressed for problems where evaluation is time consuming and the total number of analyses has to be limited around 10^3 . This aspect is of major interest, because papers on the topic such as in Karaboga and Basturk (2008) and in Karaboga and Akay (2009) often consider more than 10^5 function evaluations and therefore their results are not directly applicable in the context presented.

A rapid review of ABC is presented in Chapter 2, afterward Chapter 3 is completely dedicated to explain the technologies implemented in AsBeC algorithm. Different kind of suitable parallelization strategies are examined and depicted in Chapter 4. As validation, in Chapter 5 the new algorithm performance is accurately evaluated and compared to ABC through extensive simulations on many analytical benchmark functions. To conclude, standard settings for the new technologies and AsBeC guidelines are suggested both for serial and parallel versions, in order to preserve the original ABC simplicity and user-friendly qualities. Additionally, tuned settings for specific problems are discussed. Conclusions and further work on the topic are presented in Chapter 6.

2 The original *Artificial Bee Colony* algorithm

The *Artificial Bee Colony* (ABC) algorithm was firstly developed by Karaboga in 2005. The algorithm reproduces the behaviour of a honey bee colony searching the best nectar source into a target area. Some bees (employees) are each assigned to a food source and search the space near it. Then they come back to the hive and communicate through a dance (Frish 1967) the position of the best food sources found to other bees (onlookers), that help the first ones in the most promising regions. Nectar sources that reveal themselves non-productive are abandoned in place of eventual new fruitful positions. They are investigated by a travelling bee (scout) in the whole target area. In optimization context food sources represents input configurations and the comparisons among them is based on the objective function to optimize; non-productive food sources represent configuration not improved for some time (*limit* parameter). An entire optimization sequence composed by employee group, onlooker group and scout is referred to as an optimization *cycle*. The ABC pseudo-code is reminded below in Algorithm 1 while more details will be given in the next Chapter.

ABC algorithm tries to balance exploration and exploitation attitudes, offering worthy global and local search skills at once. As result, if compared with other competitive methods, ABC demonstrates high-quality, robustness and flexibility for a great variety of optimization problems. To enter in details consult Karaboga (2007a) and for extensive simulation comparison refer to Karaboga and Akay (2009). The main qualities of the algorithm that have been recognized (Rao et al. 2010; Singh 2009; Sharma and Pant 2011) are the following.

- Simple and easy to implement
- It can be parallelized

Algorithm 1 The ABC pseudo-code

```

1: Produce the food sources with new random configuration in the search area
2: Evaluate the quality of food sources
3: Assign each food source to a different employee
4: repeat
5:   for all employees do
6:     Produce new configuration near its food source
7:     Evaluate the quality of configuration
8:     if it is better than current employee's food source then
9:       update food source
10:    end if
11:  end for
12:  Assign each onlooker to one of the food sources, depending on their quality
13:  for all onlookers do
14:    Produce new configuration near its food source
15:    Evaluate the quality of configuration
16:    if it is better than current onlooker's food source then
17:      update food source
18:    end if
19:  end for
20:  if a new best overall food source has been found then
21:    Memorize the best food source found so far
22:  end if
23:  if a food source has not improved for some time (limit) then
24:    Replace this food source with a new random configuration in the search area (scout)
25:  end if
26: until requirements are met

```

- It can be hybridized
- It needs few control parameters
- It is flexible and robust to wide range of problems

While the deficiencies can be outlined in:

- No exploitation of the history of points analysed
- Local search and refinement skills are less efficient with respect to global search attitude

Since the original paper by Karaboga (2005) many researches on the topic were developed. In particular, many scientists focused on the goal to improve the original code and to parallelize it in an efficient way for a wide variety of disciplines. Specific modifications in all the different phases of the algorithm have shown improvements in the quality of results. Some of these improvements will be presented and used later in this paper. A good and complete outline of these works has been presented by Bolaji et al. (2013).

Although this great amount of available literature, the authors did not find any papers in which was underlined a performance gain even with few function evaluations. This framework motivates the willingness of introducing and analysing modifications effective with small bee colonies and few *cycles*. Such modifications that allow to address the deficiencies shown above are presented in the next chapter.

3 Technologies of *Artificial super-Bee enhanced Colony* (AsBeC) algorithm

This chapter is dedicated to illustrate and motivate the convenience of implementing the technologies proposed for the *Artificial super-Bee enhanced Colony*(AsBeC) algorithm. Some of the improvements here applied exploits the basic principles brought in standard ABC by other authors, but some others introduce original ideas.

The technologies presented try to speed up the best solutions in their neighbourhood, without clustering the swarm and leading to premature overall convergence. In fact, the aim of this work is to improve the local search skills of original ABC (exploitation) without worsening its global attitude (exploration), especially during the first search phases. The paragraphs below describe each technology in details considering a minimization

problem without loss of generality. Then the final AsBeC pseudo-code is outlined and commented in Paragraph 3.8, also explaining the choice of the algorithm name.

The technologies proposed are not all the ones considered by the authors, but are the ones that in Chapter 5 will have relevance for the purposes studied.

3.1 Postponed hive dance

This modification give to the employee and onlooker groups additional time to independently search nectar near their food sources. In particular, the onlooker group is assigned to the food sources only after a fixed number *check* of movements of the employees. In the same way each onlooker bee will have *check* movements instead of one to move around its food source. The greedy selection on food sources is regularly applied. *Postponed hive dance technology* permits to empathize the explorative skills of employees and the exploitation skills of onlookers, giving them more time to evolve. Anyway, the number of movements between communications should be maintained small in order to not decelerate convergence speed. The modification in the code can be seen in the pseudo-code in Algorithm 2 with the two new *for* loops on employees and onlookers in row 5 and 15. Thanks to this *Postponed hive dance*, bees will have more chances to use super-bee skills as will be shown in 3.5 and 3.6.

This modification acts like enlarging the colony size, but maintaining constant the number of food sources. Hence each food source has more chances to improve and to be selected for movement by the others. In Chapter 5 will be shown that this approach is faster than simply increasing the swarm population. Referring to PSO, Gardner et al. (2012) have shown this performance worsening when particle number becomes too large, under fixed number of function evaluations.

This *Postponed hive dances* among bee groups is inspired by nature: in fact, bees would come back to the hive to communicate their food sources only after some time, when there is more probability to have collected some pollen.

3.2 Strictly biased onlooker assignment

In the ABC onlookers are assigned to food sources by a stochastic rule, assuming a certain probability p_j related to a fitness value $fit(x_j)$ of the configuration x_j of the food source j :

$$p_j = \frac{fit(x_j)}{\sum_{k=1}^{SN} fit(x_k)} \quad \forall j \in \{1, \dots, SN\}$$

where SN is the food sources number and $fit(x_j)$ is inversely proportional to the objective function $f(x_j)$. In the present work, considering a minimization problem, the fitness fit is taken (Akay and Karaboga 2010) as follows:

$$fit(x_j) = \begin{cases} \frac{1}{f(x_j)+1} & \text{if } f(x_j) \geq 0 \\ 1 - f(x_j) & \text{otherwise} \end{cases} \quad \forall j \in \{1, \dots, SN\}$$

If all $f(x_j)$ are very similar, then the fitness function above described returns almost identical values and so all probabilities p_j are alike. Hence, the previous fitness formulation is not able to distinguish solution quality in respect to the relative current amount of $f(x_j)$. In order to always strengthen exploitation, it is possible to unbalance onlooker assignation in favour of the relative best food sources re-scaling the fitness fit to a new fitness function fit^{res} :

$$fit^{res}(x_j) = \frac{fit(x_j) - \min_{k=1, \dots, SN} fit(x_k)}{\max_{k=1, \dots, SN} fit(x_k) - \min_{k=1, \dots, SN} fit(x_k)} \quad \forall j \in \{1, \dots, SN\}$$

$fit^{res}(x_j)$ is generalized and still valid even changing the fit definition.

In the original formulation, given that bees are sequentially allocated, there is a possibility to assign multiple bees to the worst food sources or, more generally, to assign only few or none onlookers to best ones. The previous scenario is highly improbable when dealing with large-size swarm, but for small-size colonies a different approach should be preferred in order to contain function evaluations. The randomness inserted in the model by the original stochastic rule can be changed to a deterministic one. Then, an integer number N_j of onlookers is associated to food sources j proportionally to its re-scaled fitness:

$$N_j = \left\lfloor \frac{ON \cdot fit^{res}(x_j)}{\sum_{k=1}^{SN} fit^{res}(x_k)} \right\rfloor \quad \forall j \in \{1, \dots, SN\}$$

where ON is the onlooker number and the operator $\lfloor a \rfloor$ returns the highest integer smaller than a . In case the number of onlookers assigned $\sum_{j=1}^{SN} N_j$ is less than the total number of onlooker ON , the difference $ON - \sum_{j=1}^{SN} N_j$ is assigned to the best food source. In practice, no bee is ever sent to the worst food source and at least one is always sent to the best one.

Notice that the *Strictly biased onlooker assignment*, composed by fitness re-scaling and deterministic rule, does not affect appreciably the global search skills of the swarm considering small number of function evaluations. In fact, exploration is essentially up to the employee movement which remains unaltered. This technology simply exploits the best food sources to accelerate convergence on local minima, at the affordable expense of disregarding the worst areas.

3.3 Multiple parameter selection

The bee movement in ABC for the food source j is based on the modification on a single parameter i , chosen randomly between all the possible ones. Another food source $k \neq j$ is chosen randomly and the new position $x_j^{new}(i)$ for the bee associated to the food source j is:

$$x_j^{new}(i) = x_j(i) + rand[-1, 1] \cdot (x_j(i) - x_k(i)) \quad (1)$$

Where $rand[-1, 1]$ is a uniformly distributed random number in the range $[-1, 1]$. This movement tends in average to lead bees towards the best search areas, where objective function value is higher, auto-adapting the search step.

On the other hand, when dealing with high dimensionality or problems where exploration is fundamental the single parameter selection could be slow, especially if you want to obtain the best gain with few function evaluations. This is due to the fact that at the first stage of optimization exploring the space is more important. Therefore, the central idea is to promote initial exploration. Then, multiple simultaneous parameter i are chosen randomly and for each one the movement described in Equation (1) is performed. This results on average in a longer step, in terms of Euclidian distance. The number of parameters to change can be also random: in the present work is adopted a uniform probability distribution between 1 and the total number of parameters. This modification was already introduced by Akay and Karaboga (2010) in the MABC. It is also possible to adapt the probability distribution during the optimization process as will be discussed in Chapter 5.

3.4 Smart scout repositioning

The standard ABC scout consists is a random reposition of a food source in the whole initial range. This kind of placement has two main limitations:

1. It introduces consistent variability in bee movements that may deteriorate swarm efficiency. Global randomness is exacerbated in spite of the evolution and therefore exploration seems to be overrated. As a result, global colony behaviour could decline to ineffectiveness and disorganization;

2. When considering problems which bounds are unknown, the initial range represents only a reference region where the minimum is supposed to lie. In general, this kind of problems could not find a minimum inside the initial boundaries. In this situation enlarge the starting ranges to comprise the entire virtual feasible domain will slow down algorithm convergence due to the immensity of multi-dimensional space. Besides, large regions of the domain could remain unpopulated because of objective function discontinuity or not trustworthy response. Therefore a small initial search range targeted on reliable zones has to be preferred.

Both cases can be avoided following the colony best nectar positions, even out of initial boundaries. Search efforts should be concentrated in a progressively smaller domain volume whilst local minima are put apart and exploitation overcomes exploration. *Smart scout repositioning* technology places the scouts randomly in the convex hyper volume delimited by food sources, enlarged adaptively when the diameter of such hyper volume shrinks too much. For each parameter i the diameter on such dimension $diam(i)$ is defined as:

$$diam(i) = \max_{k=1, \dots, SN} (x_k(i)) - \min_{k=1, \dots, SN} (x_k(i)) \quad \forall i \in \{1, \dots, m\}$$

where m is the total number of parameters and the new position $x_j^{new}(i)$ for food source j on parameter i will be:

$$x_j^{new}(i) = \sum_{k=1}^{SN} w_k \cdot x_k(i) + \frac{rand[-1, 1]}{4} \cdot |range(i) - diam(i)| \quad \forall i \in \{1, \dots, m\}$$

where w_k are normalized random weights, $range(i)$ is the original range length for each parameter i , defined as difference between its upper and lower bound, and the operator $|\cdot|$ represents the absolute value of its argument. In other words, if $diam(i)$ is comparable with the initial range no enlargement is adopted, because the swarm is already too sparse. Otherwise, if $diam(i)$ is small compared to the initial range then the hyper volume is re-enlarged. In the present work the minimum hypercube volume is $\frac{1}{2}^m$ times the initial one, where m is problem dimensionality. The specific value of $\frac{1}{2}$ has been selected by sensitivity analyses in order not to congregate the swarm excessively.

Nonetheless, the *smart scout repositioning* described has the clear disadvantage to worsen the global swarm explorative skills, especially when in general all the bees are close to each other. This is damaging if the swarm focuses wrongly around a remote local minima, so this kind of repositioning has to be excluded when objective function presents several local minima very distant in space. Further discussion on this topic is deferred to Chapter 5.

3.5 Opposition principle

The concept of opposition based learning is widespread in the optimization field (Tizhoosh 2005), it has already been suggested in literature for the ABC algorithm (I-ABC by Sharma and Pant 2011 and OABC by El-Abd 2011) and it was recently applied in turbomachinery (Yang et al. 2013). If a random movement does not produce an improvement it is possible to try the opposite movement, with the same module and direction but reverse pointing. Given the previous bee movement for the food source j from $x_j(i)$ to $x_j^{new}(i)$ in Equation (1) the new position to test would be:

$$x_j^{opposite}(i) = 2 \cdot x_j(i) - x_j^{new}(i) \quad \forall i \in \{1, \dots, m\}$$

If the objective function $f(x_j^{opposite})$ is better than $f(x_j^{new})$ then the food source j is updated to the opposite position.

In practice, *Opposite principle* is a linear local estimator. Every function is approximately linear in its closest neighbour for a variation that goes to zero and therefore the success probability related with this modification is inversely proportional to the step module. In this paper the opposite movement is applied whenever one bee does not improve its nectar with a random movement, following the belief that it is convenient to

penalize failures offering at once a possibly better alternative. Note that this technology implies a transitory break in improvement communication, since the required bee tries the exact opposite point related to its food source independently from what the other bees have done.

Employee and onlooker groups forget their previous positions when they come back to the hive, so their first movements are always pseudo-random by Equation (1). Therefore, the only possibility to apply the modification under exam without changing the structure of the original ABC is when at least 2 onlooker bees are assigned to the same food source: if the first onlooker does not improve the food source by $x_j^{new}(i)$, then the second onlooker uses a *super-Bee* skill (see Paragraph 3.8) to perform $x_j^{opposite}(i)$. For other onlookers assigned to the same food source the procedure is repeated starting from a different random movement each time and adjourning the food source j when it improves. When *Postponed hive dance* technology is active, all the onlookers performs *check* steps keeping memory of what they are doing.

The same procedure applies to employee group, but in this case food source allocation is never multiple. This means that *Postponed hive dance* becomes necessary to make opposition available for employees.

3.6 Second order interpolation

The *second order interpolation* estimates the local and directional curvature of the objective function, acting as a second order optimization method with partial Hessian computation. This technology has to be implemented together with the *Opposite principle* (Paragraph 3.5), representing one of its possible evolutions.

The multi-dimensional parabola passing through three previous points (starting food source point $x_j(i)$, first random movement $x_j^{new}(i)$ and opposite position $x_j^{opposite}(i)$) is calculated through a linear system:

$$\begin{bmatrix} x_j(i)^2 & x_j(i) & 1 \\ x_j^{new}(i)^2 & x_j^{new}(i) & 1 \\ x_j^{opposite}(i)^2 & x_j^{opposite}(i) & 1 \end{bmatrix} \cdot \begin{bmatrix} a_i \\ b_i \\ c_i \end{bmatrix} = \begin{bmatrix} f(x_j(i)) \\ f(x_j^{new}(i)) \\ f(x_j^{opposite}(i)) \end{bmatrix} \quad \forall i \in \{1, \dots, m\}$$

and its minimum $x_j^{interpolation}(i) = \frac{-b_i}{2a_i} \quad \forall i \in \{1, \dots, m\}$ is computed and then evaluated.

Opposite principle and *Second order interpolation* have to be performed in excluding sequence: if the first random movement is improving, none of the two following steps is carried out; if the opposite movement is improving, parabola will not be estimated; if both the previous two movements does not improve the food source quality then the *Second order interpolation* is computed. This means that the food source is isolated for two rounds from the possible improvements achieved by the rest of the swarm and this could be a cause of speeding down. Consequently, it is up to this work to show that *Opposite principle* and *Second order interpolation* are generally convenient.

Moreover, as for the opposite principle, the *Second order interpolation* can be effective in ABC original structure only when at least 3 onlooker bees are assigned to the same food source. When *Postponed hive dance* technology is active with $check \geq 3$ also employees can use the *Second order interpolation* strategy.

3.7 Prophet

This hybridization, referred to as the *Prophet*, exploits the knowledge about the history of the best solutions found and how the parameters changed, the so called *optimal path*. Given the last best objective function values and its relative parameters, the algorithm guesses a new point to try, computing a weighted average of the last directions of improvement. Then the *Prophet* function is a trend predictor, independent from the optimization method used.

More in detail let $x_t(i), i = 1, \dots, m, t = 1, \dots, n$ be the vectors of the last best n improvement solution configurations, where the last optimum is indexed by n . Let $f(x_t)$ be objective function values corresponding to the configuration x_t . A normalized weight $w(t)$ is assigned to the last $n - 1$ improving movements:

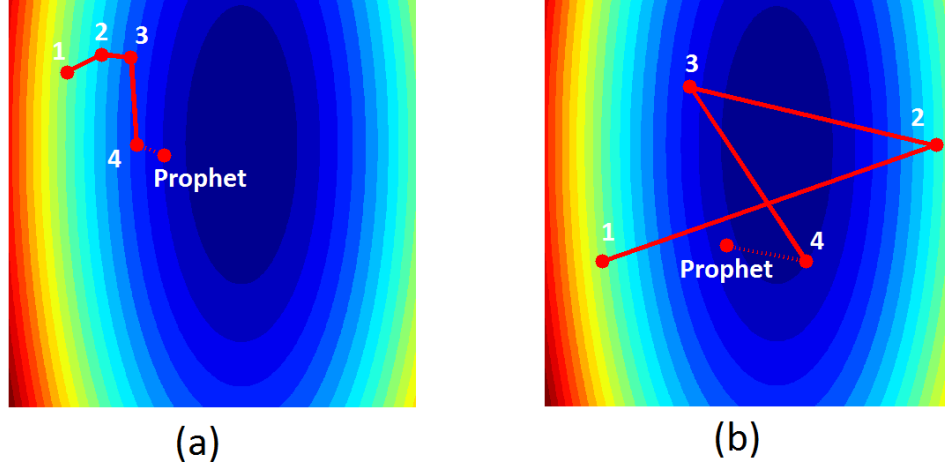


Fig. 1 A ordered optimal path (a) and chaotic optimal path (b) with their *Prophet* guess

$$w(t) = \frac{(w_t \cdot [f(x_t) - f(x_{t-1})])}{\sum_{t=2}^n w_t \cdot [f(x_t) - f(x_{t-1})]} \quad \forall t \in \{2, \dots, n\}$$

where w_t is a nonlinear function of time, chosen monotonically increasing to give more weight to the last improving steps. Then such weights $w(t)$ depends both on the quality of the improvements $[f(x_t) - f(x_{t-1})]$ and on the proximity w_t to the current solution. The *Prophet* configuration is given by a movement from the last best parameters x_n and it is computed through:

$$x^{new}(i) = x_n(i) + C \cdot \sum_{t=2}^n w(t) \cdot [x_t(i) - x_{t-1}(i)] \quad \forall i \in \{1, \dots, m\}$$

where C is a constant called *catalyst* that, found the possible direction of improvement, gives the magnitude of the step. In this paper the memory n is set to 4 and three values for the *catalyst* (0.5, 3 and 5) are taken into account. High values of the *catalyst*, $C \approx 5$, are recommended for problems in which the optimal path is not chaotic and then steps are considered reliable, such as in walk-based algorithms. On the other hand, when the optimal path is mostly disordered $C < 1$ yields to smaller movement around the best solution limiting fruitless function evaluations. This is the case of population-based algorithms, like genetic or swarm intelligence, especially during the first explorative phase.

To show the *Prophet* operability a toy example is presented in Figure 1, consisting in a bi-dimensional function as objective to be minimized. Colour map indicates diminishing values from red to blue. Two different optimal paths of length $n = 4$ on the same optimization problem are depicted and the respective *Prophet* extrapolations are reported. *Catalyst* value is maintained equal ($C = 1$) in both cases. It is easy to note that the plot on the left (a) represents a well correlated optimal path, while the one on the right (b) a chaotic one. *Prophet* response reflects these conditions with good guess for (a) and vice versa for (b), supporting the choice of bigger C for (a) and smaller for (b).

3.8 AsBeC

The technologies already presented can be classified into two main groups, outlined below, that explain the name of the new algorithm: *Artificial super-Bee enhanced Colony* (AsBeC).

1. **Enhancements.** These are modifications that do not alter the architecture of the original ABC, but make it work in a slight different way to match specific goals, such as improve the velocity on the short optimization period. Each squad of bees can have more time to evolve their nectar sources (*Postponed hive dance*); exploration can be privileged setting more than one parameter to change (*Multiple parameter selection*); for small swarms, the exploitation of the best food sources can be privileged, penalizing always the worst ones (*Strictly biased onlooker assignment*); the scout can be relocated in a range that depends on the position of the food sources (*Smart scout repositioning*);
2. **Hybridizations: *super-bee* concept.** These technologies alter the original pseudo-random movement of the bees given in Equation (1), trying to accelerate the optimization process and its accuracy. The local behaviour of the objective function can be estimated (*Opposite principle*, *Second order interpolation*) and the data history can be used to make a prediction of the next best search direction (*Prophet*). Therefore with these modification a bee assumes new abilities and it will be called a *super-bee*.

The implementation of all the technologies combined leads to the following final complete pseudo-code in Algorithm 2 for the innovative AsBeC algorithm, evolved from original ABC one.

Algorithm 2 The AsBeC pseudo-code with all the technologies activated

```

1: Produce the food sources with new random configuration in the search area
2: Evaluate the quality of food sources
3: Assign each food source to a different employee
4: repeat
5:   for 1 to check do
6:     for all employees do
7:       Produce new configuration near its food source (pseudo-random/opposite/interpolation)
8:       Evaluate the quality of configuration
9:       if it is better than current employee's food source then
10:        update food source
11:      end if
12:    end for
13:  end for
14:  Assign each onlooker to one of the food sources, depending strictly on their quality
15:  for 1 to check do
16:    for all onlookers do
17:      Produce new configuration near its food source (pseudo-random/opposite/interpolation)
18:      Evaluate the quality of configuration
19:      if it is better than current onlooker's food source then
20:        update food source
21:      end if
22:    end for
23:  end for
24:  if a new best overall food source has been found then
25:    Memorize the best food source found so far
26:    Replace this food source with a new configuration by the Prophet
27:  end if
28:  if a food source has not improved for some time (limit) then
29:    Replace this food source with a new configuration depending on the others (smart scout)
30:  end if
31: until requirements are met

```

4 Parallel Implementations

It is now well established that the future of computing is oriented on parallelized architectures. They reveal themselves very convenient at running multiple and independent resource-consuming analyses. Today every desktop computer offers several cores and therefore it is straightforward to take advantage of this technology

even without using CPU clusters or cloud computing. The clear benefit is that the same number of function evaluations can be carried out in up to as many times less as are the number of physical cores. As a consequence, the serial AsBeC code have been modified into parallel versions.

In the following subchapters three possible parallelization of bee-colony based algorithm are considered. They have already been presented by Subotic et al. (2011), but in this paper they will be implemented together with the technologies introduced in Chapter 2. In this paper the number of employees and onlookers is taken equal to 8 also because simulations have been performed on an Intel® Xeon® 8-core CPU based machine.

In presenting the following algorithms it will be assumed that the time spent for evaluating a function is roughly constant and therefore there are no substantial computational differences among possible configuration solutions.

4.1 Multi-Start Parallel approach

The Multi-Start approach (Subotic et al. 2011) is the simplest way to take advantage from availability of multiple cores. It consists in running simultaneously many independent instances of the optimization process, with different random seeds. During the optimization procedure the best result among all is taken. If p instances are run on p cores and i is the total number of *cycles* allowed in the process, then only i/p *cycles* can be completed for each instance of the serial algorithm. On the other hand, running i *cycles* for each instance means to maintain constant the total machine time, if parallelization do not slow down function evaluation times. In this ideal case, the multi-start parallel approach cannot worsen the averaged serial performance by definition.

4.2 Multi-Swarm Parallel approach

The Multi-Swarm approach is intended to better exploit the Multi-Start parallel approach considering the same number of total function evaluations. Multi-Swarm comprises communication among the different colony that are running in parallel. In particular, the authors have implemented for AsBeC the algorithm proposed by (Subotic et al. 2011). The number of colonies has to be equal to the number of food sources. Every k *cycles* each colony communicates its best food source to all the others. Then each colony restarts the optimization using as initial food sources the best food sources obtained from all the other colonies and its best one.

The number of *cycles* k has to be accurately tuned. Too small k prevents each swarm to evolve but too large k slows down the algorithm. An experimental sensitivity study conducted by the authors for AsBeC has shown that the best k value is around 20.

4.3 Bee-by-Bee Parallel approach

In the Bee-by-Bee approach (BbB) half the colony (the employee or the onlooker group) all together makes a single movement in parallel. Then it cannot be parallelized with more threads than the actual size of half the colony, so for the case under study this will be considered as 8. Therefore in every *cycle* the loops of employees and onlookers (line 6 and 16 of the pseudo-code of Algorithm 2) consist of eight parallel function evaluations.

To be precise there is a slight difference between our implementation of ABC and the original one, not specified in the pseudo-code in Algorithm 1: the food source position given by the scout bee is evaluated together with the first following employees group. This has been done in order to maintain constant the size of blocks to be parallelized. Widespread sensibility analyses have been conducted by the authors proving that this modification does not alter the original ABC performance. An equivalent approach have been pursued in AsBeC for *Prophet* evaluation.

Parallelization implies a resulting performance up to 8 times faster than in serial case for the same number of function evaluations in a 8-core machine. Furthermore it permits to complete up to 8 times the function evaluations in the same machine time. Losses in performance are expected since during the 8 parallel runs

Function name	Properties	Dimensions	Range for each dimension
Sphere	U S	50	$-100 < x_i < 100 \quad \forall i$
Dixon Price	U N	20	$-10 < x_i < 10 \quad \forall i$
Schwefel	M S	5	$-500 < x_i < 500 \quad \forall i$
Styblinski Tang with noise (15%)	M S	5	$-5 < x_i < 5 \quad \forall i$
Levy	M S	10	$-100 < x_i < 100 \quad \forall i$
Rastrigin	M S	10	$-10 < x_i < 10 \quad \forall i$
Perm	M N	5	$-5 < x_i < 5 \quad \forall i$
Rosenbrock	M N	10	$-5 < x_i < 5 \quad \forall i$
Ackley	M N	10	$-20 < x_i < 70 \quad \forall i$
Griewank	M N	30	$-600 < x_i < 600 \quad \forall i$

Table 1 The benchmark function set. U=unimodal; M=multimodal; S=separable; N=non-separable;

there is no improvement communication and no sequential adjourning of upgraded food sources. Therefore, the colony convergence slows down but its explorative skills are intensified. This approach is affordable when time bottleneck are not in threads communication, that are very frequent, but in function evaluation as in the present case.

5 Numerical Experiments on Test Function Benchmark

In order to compare AsBeC technologies with respect to ABC a set of 10 analytical mathematical test functions have been selected as a benchmark (Table 1). Even if this set is far from represent a good sample of real-world numerical optimizations, it tries to gather many characteristics that appear in engineering problems. In fact, the set contains unimodal, multimodal, separable and not-separable functions with domain dimensions between 5 and 50. Moreover it contains functions with few far local minima, thousands of closed local minima, stochastic noise and very narrow holes. For a complete description of the functions chosen refer to Yang (2010). All the functions proposed are modified in order to present a unique global minimum equal to 0, with a numerical tolerance set to 10^{-16} by the authors.

The authors highlight that the aim of the work is not to find the best configuration for all the real-like problems: the goal is to give evidences that the modifications proposed in Chapter 3 and 4 can improve the performances on the benchmark set and then presumably they are a better starting point for engineering problems. Additionally, will be provided evidences when to use some technology and when not.

The analysis are done in the following way: for each test function and for each configuration of technologies 300 runs with a colony of 16 bees are performed with *limit* parameter equal to 10 and a maximum of 1600 function evaluations, corresponding to 100 overall *cycles*. The choice of 300 runs is made to remove the incidence of the random component in the algorithm. Since the total number of function evaluations is low the results are far from convergence and thus the variance is higher than in other experiments such as in Karaboga and Akay (2009). Only one scout bee per *cycle* is admitted and so the *limit* parameter is set smaller than advised for original ABC, $m \cdot ON$ or $m \cdot ON/4$ (Karaboga and Bahriye 2007b); in fact, the scout bee has to be activated sufficiently often in order to assess the effects of *Smart scout repositioning* technology. A performance estimator of a combination of technologies with respect to the standard ABC is the gain G defined as:

$$G(conf, funct, FE) := \frac{M_{rep}(f_{best}(ABC, funct, FE, rep))}{M_{rep}(f_{best}(conf, funct, FE, rep))} \quad (2)$$

where *conf* is the configuration of technologies, *rep* is the run repetition, *FE* is the number of function evaluations performed, *funct* is the benchmark function selected, f_{best} is the best objective value obtained and M_{rep} represents the median operator over repetitions. Since all the benchmark functions have minimum value equal to 0, the gain G is proportional to the quality of the median performance of the algorithm. Starting from Equation (2) it is possible to define a *Mean Logarithmic Gain* (MLG) over all the benchmark functions, representing

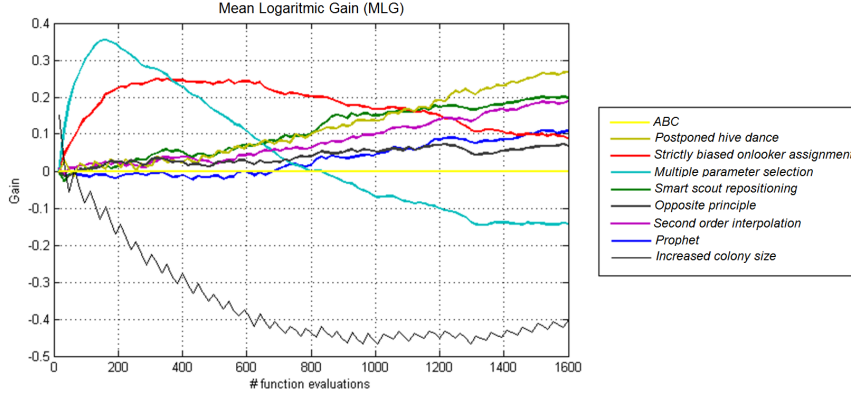


Fig. 2 *MLG* for single AsBeC technologies activated with respect to ABC

the median performance of the algorithm in terms of order of magnitude gained, averaged over the 10 function tested:

$$MLG(conf, FE) := \frac{1}{10} \cdot \sum_{funct=1}^{10} \log(G(conf, funct, FE))$$

A *MLG* of 0 represents a null mean gain of orders of magnitude with respect to the ABC, while a positive value represents a positive average between order gained and (possibly) lost.

Since in parallel cases it is possible to run simultaneously multiple evaluations, a Gain \hat{G} and Mean Logarithmic Gain \widehat{MLG} function of time t instead of function evaluation FE are also introduced.

The analyses are performed using MATLAB[®] and the results are reported in the following sections. A key note is that the test functions used are analytical and therefore the computational time is fast, differently from the framework studied. In order to present valid results in parallel versions, the authors have taken into account the number of function evaluations and then it is supposed that the whole machine time is spent in evaluate the functions and not in creating threads and make them communicating. Moreover each evaluation time has been considered as constant in all the test functions.

5.1 Test on serial configurations

First of all, it is crucial to evaluate the performance of each single technology proposed and Figure 2 illustrates their *MLG*. All the technologies except one represent small improvements. The only technology that worsen the ABC is the *Multiple parameter selection*, but only after about 800 function evaluations. In fact, moving multiple parameters helps the exploration at the beginning, while it is then unlikely to refine the best solutions moving many parameters at once and so exploitation is reduced. Notice also the big impact of *Strictly biased onlooker* assignment technology, especially at the beginning when the ABC assignment by stochastic rule could lead to waste resource to exploit poor regions.

In addition, the effect of triple the colony size from 16 to 48 in place to apply *Postponed hive dance* with $check = 3$ has been studied. As introduced in Paragraph 3.1, the Figure 2 evidences the very poor behaviour of increasing the colony size, establishing that *Postponed hive dance* technology results much faster at least with few function evaluations.

Then all the possible combination of technologies were tested in order to capture all the interactions between them. A statistical analysis on the data allows to select the best combination among the dominating solutions. A configuration is called dominating if on all the functions considered it has shown better medium results respect with ABC. All the combinations were compared gathering the performance as done in Figure 2

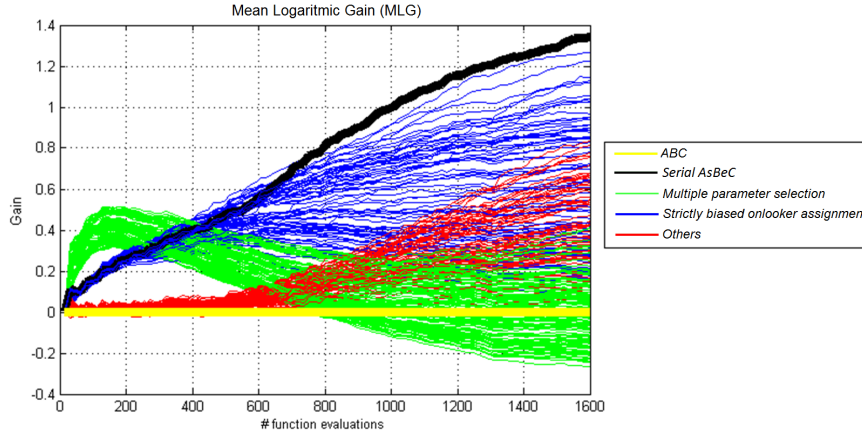


Fig. 3 *MLG* for all the AsBeC technologies combinations with respect to ABC

and the results are shown in Figure 3. Notice that there are mainly three clear streams of combinations: the ones with *Multiple parameter selection* (green), the ones with *Strictly biased onlooker assignment* (blue) and all the others (red). The best configuration (black) was chosen to be the dominating one with the highest *MLG* after 1000 functions evaluations. The best configuration corresponds to the following technologies activated:

- *Postponed hive dance* with *check* = 3
- *Opposition principle* and *Second order interpolation*
- *Strictly biased onlooker assignment*
- *Prophet* with *catalyst* = 0.5

although other combinations of technologies were rated very similar. The previous technologies selected defines what will be called the serial AsBeC algorithm.

Some qualitative comments can be done. As already observed in Figure 2 if one want to optimize in very few cycles (10 – 20, corresponding to 10^2 function evaluations) and the starting point is far from the optimum, *Multiple parameter selection* could be privileged. Even if this suggest to switch from a multiple parameter selection to single parameter selection after some time using an adaptive technology this does not advantage the final performance of the algorithm with 10^3 function evaluations. This fact has been proven by the authors; it can be explained by the fact that food sources are usually far at the switch and exploiting capability is slowed.

However, *Multiple parameter selection* is recommended also when dimensionality is very high or in general when the specific optimization problem requires a very strong explorative skills. These two are the cases of 50 dimensions Sphere, 30 dimensions Griewank or Ackley function herein tested, for which the authors verified excellent performance of this technology in the entire simulation schedule. A further experiment has been conducted increasing the dimensionality of all the 10 benchmark functions to 50 dimensions. In this case *Multiple parameter selection* reveals itself as the most proficient technology in terms of *MLG*.

The *Smart scout repositioning* technology was not selected in the best configuration even if its activation brought additional benefits in the majority part of the functions. This is due to the fact that where wide exploration is fundamental (Styblinski Tang with noise and Ackley functions), the activation of this technology performs worse than standard ABC. Whenever the problem does not present these peculiarities it could be beneficial to activate *Smart scout repositioning* technology.

The serial AsBeC algorithm selected performs on the benchmark set as shown in Figure 4. For each function the median and mean among all the repetition of the objective function has been plot. The standard deviation was not inserted in the graph for ease of visualization, but it is in lower for AsBeC accordingly with its better performance. Many peculiar trend in the graphs (such as the one in Ackley function) can be easily explained by studying the shape of the function itself. Notice that the results are not converged and in many functions

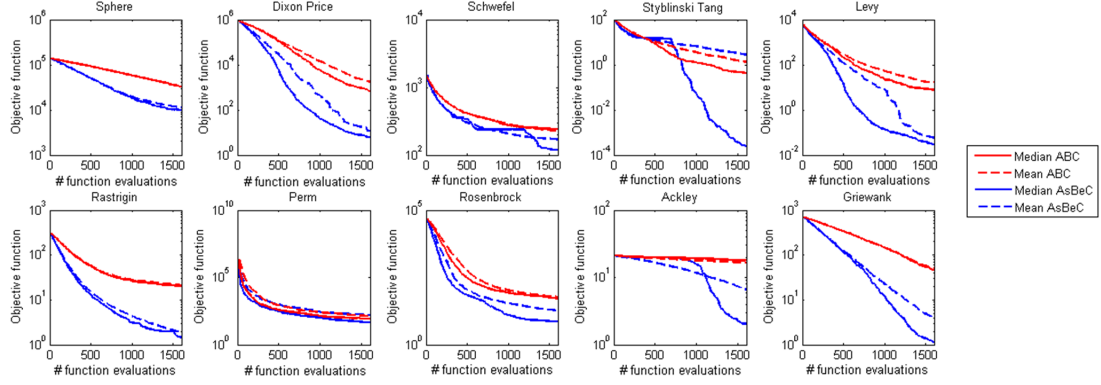


Fig. 4 The results for the serial AsBeC and the ABC on the test functions. Median and mean of objective values are shown

are still far for the global optimum. Anyway the comparison is still solid, since in many real-like problems the optimum is not known and designers are interested in obtaining the best possible solution in a fixed amount of time.

5.2 Test on Multi-Start approach

As explained in Paragraph 5.1, the Multi-Start can be run in parallel without great loss of time on as many cores as the hardware availability allows. The following results are presented on 8 cores.

If the total number of function evaluations has to be taken small the results are expected to be poor since only 200 function evaluations (corresponding to 13 cycles) are possible for each colony run in parallel. Figure 5 on the left (a) shows worse performance for any Multi-Start in respect with any serial approach (Figure 3).

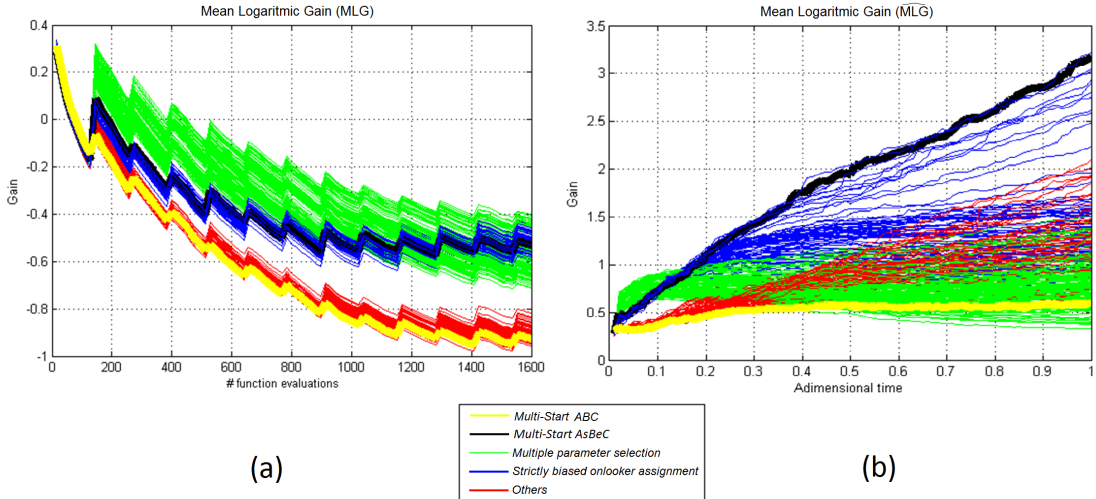


Fig. 5 Mean Logarithmic Gain for the Multi-Start AsBeC with respect to the ABC in terms of functions evaluations (a) and in terms of adimensional machine time (b)

Much more interesting is the case where 8 colonies are run in parallel for 100 *cycles* and the best is compared through \widehat{MLG} , hence allowing to execute up to 12800 function evaluation (i.e. $1600 \cdot 8$). The results are shown in Figure 5 on the right (b). The gain in respect to the serial ABC is calculated to the base of time elapsed, converted in adimensional form between 0 and 1. The best technology configuration chosen is still the same of serial AsBeC and will be called AsBeC Multi-Start. This is expected because Multi-Start is a simple repetition of exactly the same serial algorithm on the same total number of function evaluations. The ABC Multi-Start gains up to ≈ 3 times the performance of the ABC, while the AsBeC case is up to ≈ 1000 ($\widehat{MLG} \approx 3$) times better. This means that Multi-Start approach is much more useful in AsBeC.

In conclusion, this parallel approach is unsuccessful when considering the same total number of function evaluation as in serial case, while it is very useful when each colony performs that number of function evaluations. This is because Multi-Start approach needs some time to properly evolve each serial optimization.

The results of the selected configuration (Multi-Start AsBeC) on the single functions are then shown in Figures 8 and 9 in the following Paragraph 5.5.

5.3 Test on Multi-Swarm approach

Multi-Swarm approach amplifies the Multi-Start advantages as it is evident in Figure 6. Since communication among the 8 swarms occurs every 17 *cycles*, the MLG is exactly the same of Figure 5 (a), because no communication is possible within 13 *cycles*. Therefore, all the observations done about the poor behaviour when total number of function evaluation is maintained constant are valid for Multi-Swarm too.

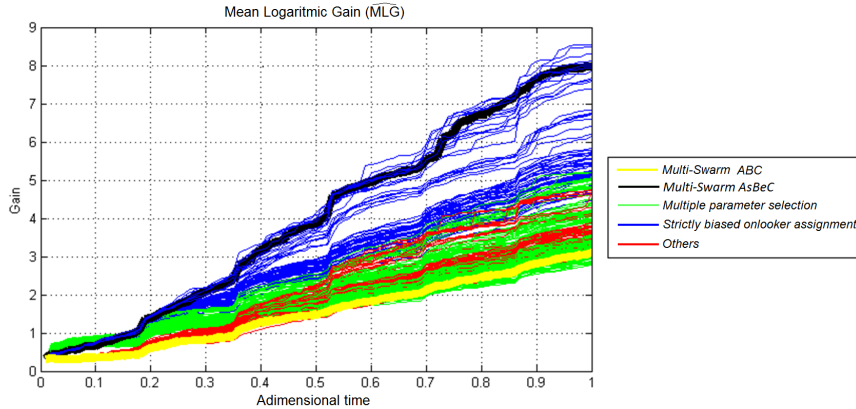


Fig. 6 Mean Logarithmic Gain for the Multi-Swarm AsBeC (communications every 20 *cycles*) with respect to the adimensional time of the serial ABC

On the other hand, Multi-Swarm \widehat{MLG} is impressive selecting the same technology combination of serial and Multi-Start AsBeC: the new defined Multi-Swarm AsBeC gains ≈ 8 order of magnitude over serial ABC at 12800 function evaluations and then it almost triples Multi-Start performance. Instead, Multi-Swarm ABC is ≈ 1000 times (3 orders of magnitude) better than serial ABC.

Figure 6 reveals the presence of other technology combinations that perform a little better than the selected one, but they are not dominating the serial ABC. Therefore the selected configuration remains unaltered because the final aim of the present work is to identify a reference AsBeC algorithm which reveals itself high-performing in its serial or parallel approaches.

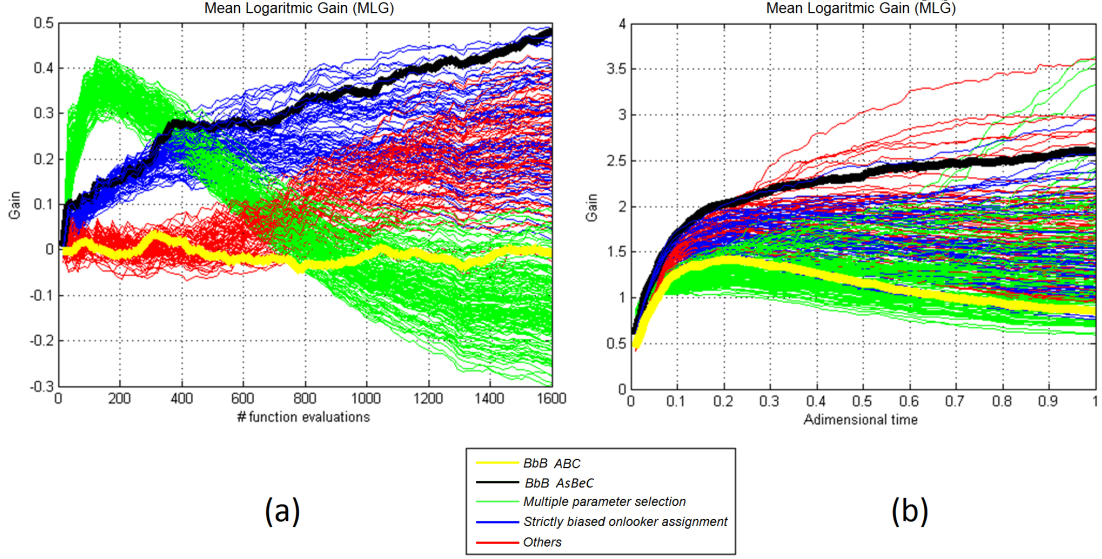


Fig. 7 Mean Logarithmic Gain for the BbB AsBeC with respect to ABC in terms of functions evaluations (a) and adimensional machine time (b)

5.4 Test on Bee-by-Bee approach

The same procedure as in 5.2 and 5.3 is followed for the Bee-by-Bee approach (BbB). In Figure 7 is presented the analogous of Figure 5. First of all, in this case MLG is worse in respect to serial AsBeC, due to the deficiencies explained in Paragraph 4.3. However, should be taken into account that the total time of execution is reduced up to 1/8 of the serial one. Moreover, BbB approach is much better in respect to Multi-Start/Multi-Swarm when total number of function evaluations is preserved.

In order to exploit the parallel approach maximizing results quality you must reason in terms of \widehat{MLG} (Figure 7 (b) on the right). Selecting the same technology configuration as before we define the BbB AsBeC algorithm, whose \widehat{MLG} is always higher than its respective serial. Nevertheless, none of the combinations dominates the standard ABC even if the selected one is surpassed only in function 3 by a very small value (Figure 9). As can be seen in Figure 7 on the right (b) there are some technology combinations with higher \widehat{MLG} , but only after 1/4 of the time (≈ 3200 function evaluations). Some of those technologies have *Multiple parameter selection* activated or *Strictly biased onlooker assignment* deactivated. Another reason for the choice of the selected BbB AsBeC will be presented in the next paragraph.

5.5 Comparison among serial and parallel AsBeC approaches

In the previous subchapters the configuration and setting of the serial AsBeC algorithm, the Multi-Start AsBeC, the Multi-Swarm AsBeC and the BbB AsBeC were defined. Observing the performance on constant number of function evaluation bounded by $\approx 10^3$, the serial AsBeC has shown the best results. In Figure 8 the performance in terms of median objective value over 300 repetitions on all the benchmark functions of all the algorithms is presented. As already stated Multi-Swarm AsBeC in this case coincide with Multi-Start AsBeC.

The results are different when performance is measured with respect to time and it is possible to exploit multicore architecture in parallel approaches, in order to increase as much as possible the number of function evaluations performed per time unit. The best algorithm until $\approx 1/4$ of the reference time, corresponding to approximately 3200 function evaluations, is the BbB AsBeC (Figure 7 on the right), while for more available

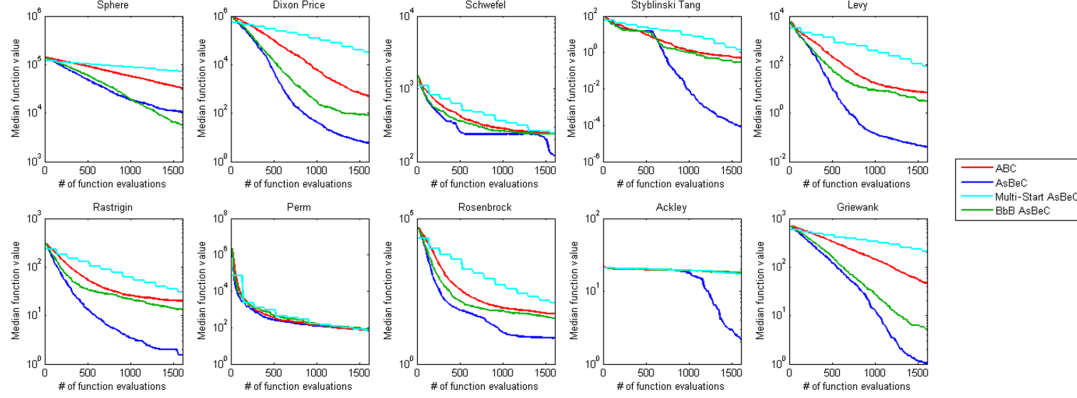


Fig. 8 The median results of the test functions for ABC, AsBeC, BbB AsBeC and Multi-Start AsBeC for constant number of function evaluations

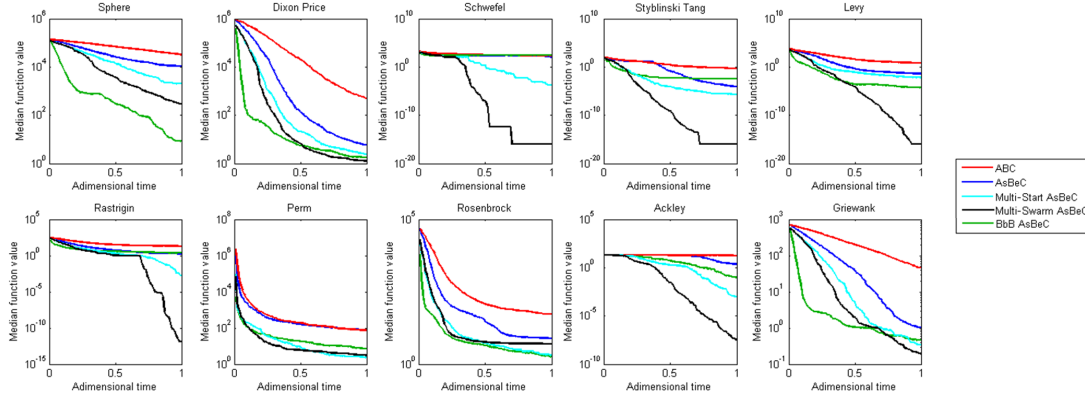


Fig. 9 The median results of the test functions for ABC, AsBeC, BbB AsBeC, Multi-Start AsBeC and Multi-Swarm AsBeC as function of adimensional time of the serial ABC

time the best results are obtained by the Multi-Swarm approach (Figure 6). Notice that specific discrimination numbers such as $1/4$ of the adimensional time depends on the specific 8-core parallelization; however, guidelines are general, so it is convenient to choose BbB AsBeC below a certain number of total function valuations, depending on hardware capability, otherwise switch to Multi-Swarm AsBeC approach. The performance of all the AsBeC algorithms are then compared for each benchmark function in Figure 9 in terms of median results.

5.6 Long run test on AsBeC

In conclusion, a test run with a total number of $5 \cdot 10^5$ function evaluations is performed, about the same as in other ABC paper (Karaboga and Basturk 2008). This has been done to see if the AsBeC selected to work well on the short period is still good. The *limit* parameter was reset to the original value proposed by Karaboga and Akay (2009), which usually leads to better performance especially when considering numerous function evaluations.

The Figure 10 illustrates the median results for ABC, AsBeC and BbB AsBeC. In 7 of the 10 test functions the AsBeC still performs better than the ABC. The clear difference that can be seen in Griewank and Styblinski

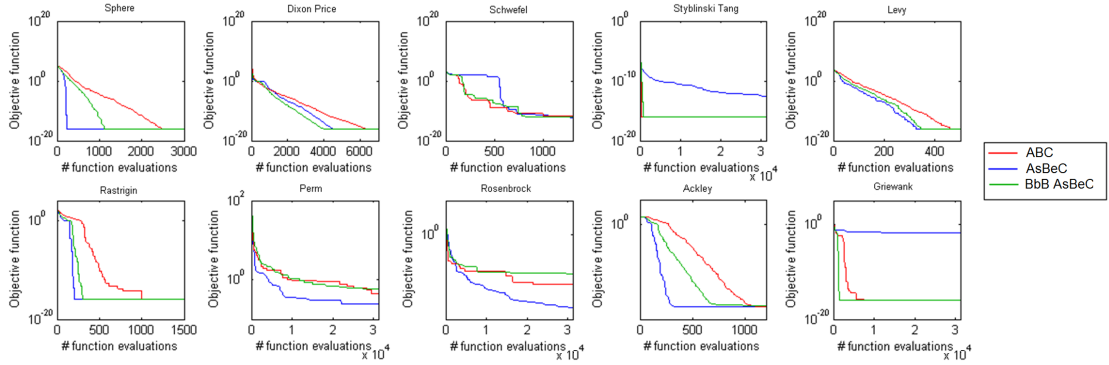


Fig. 10 The median results of the objective for ABC, AsBeC and BbB AsBeC as function of adimensional time

Tang function is due to the fact that AsBeC has less explorative skills with respect to ABC and BbB AsBeC, thus in these cases the technology combination selected should be re-considered. BbB AsBeC shows intermediate performance, according to its behaviour depicted in Paragraph 5.4 and 5.5. Notice that in almost all the functions the algorithms have reached the precision of 10^{-16} before the total evaluations allowed ($5 \cdot 10^5$).

6 Conclusions

In this paper the authors modified the ABC algorithm with the goal to better perform when function evaluations are slow and therefore their total number should be limited ($\approx 10^3$) or a multi-core architecture can be well exploited. The novel *Artificial super-Bee enhanced Colony* algorithm (AsBeC) includes some simple but effective hybridizations included in the *super-bee* principle and some enhancements in the organization of the colony behaviour. The new algorithm has shown results that outclass ABC in a great variety of analytic benchmark functions, reaching approximatively an average of 10 times more precise solutions after 1000 function evaluations.

Three parallelization strategies that take advantage from multi-core availability and therefore can perform more evaluations have been studied. Two of them (Multi-Start AsBeC and Multi-Swarm AsBeC) consist in multiple parallel colonies with and without communication abilities. The third technique (BbB AsBeC) is an internal parallelization of employees and onlookers groups, allowing to perform more *cycles* in the same total time of the serial algorithm. The first ones perform better with more function evaluations while the last one has shown impressive results in the short time. In detail, with the 8-core architecture used for the test, the BbB AsBeC has proved the best performance until the time that the serial ABC used to perform about 450 function evaluations. After that time the Multi-Swarm AsBeC reveals itself the best algorithm gaining up to 8 orders of magnitude.

Moreover major guidelines about AsBeC standard settings for generalized high-performance qualities, eventual technology activation for specific aims, serial/parallel usage and tuning have been largely discussed in order to preserve ABC intrinsic simplicity.

The present article is intended to be the first part in a wider framework. The second part will consider a real-like application to the engineering optimal design of aeronautical turbomachinery through *Computational Fluid Dynamics* analyses, environment in which the improved algorithm was originally conceived by the authors.

Acknowledgements The authors would like to thank GE Avio S.r.l. and its R&TD department, especially Ing. Francesco Bertini. Their collaboration was fundamental in the successfully application of AsBeC algorithm to turbine design.

References

- Akay B., & Karaboga, D. (2010). A modified artificial bee colony algorithm for real-parameter optimization. *Information Sciences*.
- Bertini, F., Dal Mas, L., Vassio, L., & Ampellio, E. (2013). Multidisciplinary Optimization for Gas Turbines Design. AIDAA Conference, Napoli, Italia.
- Bolaji, A., Khader, A., Al-Betar, M., & Awadallah, M. (2013). Artificial Bee Colony Algorithm, its variants and applications: a survey. *Journal of Theoretical and Applied Information Technology*, 47(2), 434-459.
- El-Abd, M. (2011). Opposition-based artificial bee colony algorithm. Proceedings of the 13th annual conference on Genetic and evolutionary computation, 109-116.
- Frish, K. (1967). *The Dance Language and Orientation of Bees*. The Belknap Press of Harvard University Press, Cambridge, Massachusetts.
- Gardner, M., McNabb, A. W., & Seppi, K. D. (2012). A speculative approach to parallelization in particle swarm optimization. *Swarm Intelligence*, 6 (2) , 77-116.
- Karaboga, D. (2005). An idea based on honey bee swarm for numerical optimization. Technical Report TR06, Erciyes University, Turkey. http://mf.erciyes.edu.tr/abc/publ/tr06_2005.pdf. Accessed November 20, 2013.
- Karaboga, D. (2007). A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. *Journal of Global Optimization*, 39, 459-471.
- Karaboga, D., & Bahriye, B. (2007). Artificial Bee Colony (ABC) Optimization Algorithm for Solving Constrained Optimization Problems. *LNCS: Advances in Soft Computing: Foundations of Fuzzy Logic and Soft Computing*, 452(9), 789-798.
- Karaboga, D., & Basturk, B. (2008). On the performance of artificial bee colony (ABC) algorithm. *Applied Soft Computing*, 8(1) , 687-697.
- Karaboga, D., & Akay, B. (2009). A comparative study of Artificial Bee Colony Algorithm. *Applied Mathematics and Computations*, 214, 108-132.
- Kennedy, J., Eberhart, R. (1995). Particle Swarm Optimization. Proceedings of IEEE International Conference on Neural Networks IV, 1942-1948.
- Price, K., Storn, R., & Lampinen, J. (2005). *Differential Evolution - A Practical Approach to Global Optimization*. Springer, Berlin.
- Rao, R. S., Narasimham, S.V.L., & Ramalingaraju, M. (2010). Optimization of distribution network configuration for loss reduction using artificial bee colony algorithm. *International Journal of Electrical Power and Energy Systems Engineering (IJEPESE)* , 1(2), 708-714.
- Sharma, T.K., & Pant, M. (2011). Enhancing the food locations in an Artificial Bee Colony algorithm. IEEE Symposium on Swarm Intelligence (SIS), 1-5.
- Singh, A. (2009). An artificial bee colony algorithm for the leaf-constrained minimum spanning tree problem. *Applied Soft Computing*, 9(2), 625-631.
- Subotic, M., Tuba, M., & Stanarevic, N. (2011). Different approaches in parallelization of the artificial bee colony algorithm. *International Journal of Mathematical Models and Methods in Applied Sciences*, 5(4), 755-762.
- Tizhoosh, H. (2005). Opposition-based learning: A new scheme for machine intelligence. Proceedings of international Conference on Computational Intelligence for Modelling, Control and Automation, CIMCA.
- Tuba, M., Bacanin, N., & Stanarevic, N. (2012). Adjusted artificial bee colony (ABC) algorithm for engineering problems. *WSEAS Transactions on Computers*, 11(4), 111-120.
- Yang, X. S. (2010). Test problems in optimization, in Eds Xin-She Yang, *Engineering Optimization: An Introduction with Metaheuristic Applications*. John Wiley & Sons.
- Yang, X., Liu, B., & Cao, Z. (2013). Opposition-Based Artificial Bee Colony Algorithm Application in Optimization of Axial Compressor Blade. ASME Turbo Expo 2013, San Antonio (TX), USA.